Golovin – AI do grania w tekstowe gry przygodowe

Bartosz Kostka, Jarosław Kwiecień, Szymon Malik

Uniwersytet Wrocławski

15 lutego 2017

Spis treści

1 Wprowadzenie

Wraz z rozwojem badań nad inteligencją obliczeniową oraz dostępnością mocy obliczeniowej uczeni podejmują, często z powodzeniem, próby rozwiązywania coraz to trudniejszych problemów. Natomiast gry komputerowe mogą dostarczyć platformy do uczenia i testowania tworzonych rozwiązań. Tak jest na przykład z, niegdyś popularnymi, przygodowymi grami tekstowymi, w których komunikacja na linii gracz-gra odbywa się w postaci tekstowej. Szeroka dostępność takich gier oraz blisko zerowe wymagania sprzętowe potrzebne do ich obsłużenia pozwalają w pełni skupić uwagę i zasoby na rozwiązywanym zadaniu.

1.1 Problem

Stworzenie w pełni autonomicznego agenta, który z powodzeniem potrafi grać w dowolną przygodową grę tekstową jest uznawany za problem "AI-zupełny". Jego rozwiązanie wiązałoby się z nastaniem tak zwanej "osobliwości", czyli sztucznej inteligencji, która dorównuje inteligencji człowieka.

Taki agent logicznie realizuje zadania, z których każde z osobna stanowi wyzwanie:

- rozumienie *znaczenia* tekstu pisanego
- gromadzenie wiedzy, wyciąganie wniosków i podejmowanie decyzji
- komunikowanie, w sposób zrozumiały, swoich akcji

1.2 Źródła i wcześniejsze prace

Tematyka ta, ze względu na swoją złożoność, dopiero zyskuje na popularności. Nie dotarliśmy do żadnych konkretnych publikacji poruszających ją bezpośrednio. Natomiast ukazuje się wiele pracy traktujących o problemach, które są jego składowymi, wspomnianymi powyżej.

Wykorzystujemy pośrednio lub bezpośrednio osiągnięcia modeli neuronowych w dziedzinie przetwarzania języka naturalnego, do jego modelowania oraz reprezentacji słów:

- word2vec - implementacja w TensorFlow:
  `https://www.tensorflow.org/tutorials/word2vec/`
- rozszerzenie o ważenie istotności widzianych słów modelu językowego zaproponowanego przez Bengio et al. w:
  `http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf`

Ponadto, do parsowania zdań i tworzenia drzew rozkładu wykorzystywane są narzędzia takie jak:

- Natural Language Toolkit – platforma wspomagająca pisanie programów używających języka naturalnego
  `http://www.nltk.org/`

- Simple Python Statistical Parser – prosty parser bazujący na algorytmie Cocke'a-Youngera-Kasamiego (CYK) uczący się probabilistycznych gramatyk bezkontekstowych (PCFG) z baz danych *QuestionBank* oraz *Penn treebanks*:
  `https://github.com/emilmont/pyStatParser`

W minionym roku po raz pierwszy odbyły się zawody botów grających w przygodowe gry tekstowe:
`http://atkrye.github.io/IEEE-CIG-Text-Adventurer-Competition/`

Zwyciężył w nich agent, który jest punktem odniesienia w testowaniu naszych rozwiązań. Jak podaje organizator, w środowisku testowym zawodów zwycięzca zdobył 18 punktów na 100 możliwych do zdobycia. Twórcami byli David Wingate, Daniel Ricks, Nancy Fulda i Ben Murdoch z laboratorium Perception Control and Cognition w Brigham Young University. Repozytorium z ich programem dostępne jest tutaj:
`https://github.com/murdo25/Competition-Repo`

2 Zastosowany algorytm

Repozytorium z programem i informacjami potrzebnymi do uruchomienia środowiska można znaleźć na GitHubie pod tym adresem: `https://github.com/Kostero/text_rpg_ai`

2.1 Modele neuronowe

Jak wspomniano we wprowadzeniu, wykorzystane zostały dwa modele neuronowe:

- **word2vec**
  Umożliwia on zanurzenie słów w przestrzeni wektorowej, ułatwia to operacje na słowach; wyrazy bliskoznaczne pojawiają się w podobnych kontekstach, stąd ich reprezentacje, jako punkty w przestrzeni wielowymiarowej, często są blisko siebie.

- **autoregresywny model językowy**
  Rozszerzenie n-gramowego modelu Bengio o możliwość wprowadzenia dowolnie długiego wejścia, gdzie nadmiarowe wektory są "uśredniane" do jednego wektora. Jedna warstw sieci uczy się wag potrzebnych do obliczenia tego dodatkowego wektora, te wagi są wykorzystywane w naszym algorytmie do ważenia "istotności" słów.

Ze względu na specyfikę gier (klimaty fantasy/rpg), oba modele były trenowane na kolekcji ponad trzech tysięcy anglojęzycznych książek z gatunku fantastyki.

2.2 Strategia rozgrywki

Ogólna zasada generowania komend

- znajdujemy wszystkie rzeczowniki w opisie miejsca (informacja o stanie gry), oraz w ekwipunku - posiadane przedmioty
- wyznaczamy ich wyrazy bliskoznaczne (sąsiednie wektory)
- w bazie komend wyszukujemy akcje zawierające te rzeczowniki, jeżeli komenda zawiera synonimy to w ich miejsce wstawiamy oryginalne słowa
- każdej komendzie przypisywana jest waga zależna od: podobieństwa synonimów do pierwotnych słów, rzadkości słów w komendzie w stosunku do słów w innych znalezionych komendach, wag słów z modelu neuronowego oraz liczby słów zgodnych ze słowami oryginalnymi
- na podstawie powyższych wag generujemy proporcjonalne do nich prawdopodobieństwa
- z tak wyznaczonego rozkładu prawdopodobieństwa nad możliwymi akcjami losujemy komendę do wykonania

Ekwipunek

- po wejściu do nowego miejsca szukamy w opisie rzeczowników, które występują rzadko i mają wysoką wagę, następnie próbujemy je podnieść
- w przypadku powodzenia - zmiana stanu ekwipunku - generujemy komendy z nowymi przedmiotami

Niepowodzenie akcji

- jeżeli komenda nie przyniesie żadnego efektu - nie zmieni stanu gry, opisu otoczenia (co zdarza się bardzo często ze względu na specyfikę agenta) - jest wpisywana na czarną listę, która przypisana jest do miejsca
- po każdej zmianie stanu gry czarne listy są czyszczone

Battle Mode

Ma największy priorytet - akcje są wykonywane przed wszystkimi innymi, jako że od nich często zależy życie bohatera gry.

- spośród komend zawierających czasowniki *attack, kill, fight, shoot, punch* wybrano około 70 najczęściej występujących
- jeżeli w opisie natrafimy na rzeczownik występujący, w którejś z tych komend to wykonujemy dla niego komendę

- powtarzamy ją kilkukrotnie (nawet w przypadku niepowodzenia) - czasem należy zaatakować przeciwnika więcej niż raz by go pokonać

Eksploracja

Chodzenie po świecie odbywa się w sposób losowy. Jest to spowodowane tym, że różne miejsca mogą mieć ten sam opis, bądź te same miejsca różne opisy, co w niektórych przypadkach może uniemożliwić stworzenie prawdziwego grafu (mapy świata).

2.3 Przygotowanie bazowych komend

Podstawowe komendy zostały wygenerowane w ramach preprocessingu. Przygotowano je w oparciu o komendy znalezione w anglojęzycznych poradnikach do gier. Niestety zwykłe komendy okazały się niewystarczające i skorzystaliśmy z tekstów ciągłych. Przy pomocy narzędzi do przetwarzania języka naturalnego oraz wyżej opisanych parserów języka angielskiego tworzyliśmy drzewa kolejnych zdań. W tych drzewach szukaliśmy poddrzew odpowiadającym prostym wyrażeniom czasownikowym. Niestety, tutaj także nie wszystkie takie wyrażenia nadawały się, dlatego na podstawie kilkuset otagowanych przez NLTK zdań, wybraliśmy typy które nas interesują i dołączyliśmy je do bazowych komend. Takie zabiegi pozwoliły nam uzyskać ponad 800 000 bazowych komend z następujących stron internetowych:

- `https://solutionarchive.com/`
- `http://www.gameboomers.com/`
- `http://www.plover.net/~davidw/sol/`

2.4 Wyniki testów

Aby porównać nasz program ze zwycięskim botem z zawodów, nie mogliśmy niestety użyć testowej gry, na której były sprawdzane programy na zawodach, jako że będzie ona użyta ponownie w tym roku i jest utrzymywana w tajemnicy.

W czasie testów korzystamy zatem z kilkunastu dostępnych gier i interfejsu do tego przeznaczonego, który jest dostępny tutaj:
`https://github.com/danielricks/textplayer`
Korzysta on z interpretera *Z-code*'u Frotz:
`https://github.com/DavidGriffith/frotz`

Gry te zawierają funkcję wyniku, która pozwala nam ocenić, jak dużo nasz program osiągnął w konkretniej grze i porównać różne podejścia.

W poniższej tabeli przedstawiono wyniki symulacji (maksymalna liczba punktów zdobytych przez agentów). Porównywany jest program stworzony przez nas ze zwycięskim botem z zawodów, któremu pozwoliliśmy działać na komputerze z 8 CPU i 16 GB RAMu przez 15 minut.

Nasz bot, zapytany o swoje imię odpowiedział: Golovin. Tak więc został nazwany.

gra	Golovin	UltimateAgent	maksimum
Advent.z5	36	36	350
Adventureland.z5	0	0	100
Chaos.z5	5	5	N/A
Murdac.z5	10	10	250
Parc.z5	0	0	400
awaken.z5	0	0	50
break-in.z5	0	0	150
bunny.z5	3	0	60
causality.z5	0	0	N/A
cavetrip.z5	5	0	500
curses.z5	2	0	550
death.z5	20	0	100
detective.z5	130	30	360
fable.z5	0	0	50
frozen.z5	0	0	N/A
karn.z5	0	0	170
lily.z5	0	0	10
loose.z5	0	0	50
minster.z5	0	0	50
night.z5	2	0	10
omniquest.z5	10	5	50
parallel.z5	0	0	150
reverb.z5	0	0	50
temple.z5	0	0	35
zork1.z5	40	5	350
zork2.z5	10	0	400
zork3.z5	1	0	7
ztuu.z5	0	0	100

3 Wnioski i możliwości rozwoju

Pomimo stosunkowo dobrych wyników - nasze rozwiązanie okazało się bardziej skuteczne od zwycięskiego agenta - wciąż stoimy na początku drogi ku skutecznemu AI w grach tekstowych.
Zaprezentowane rozwiązanie jest tak na prawdę randomizowanym algorytmem, który jest tylko „wspomagany" przez algorytmy uczenia maszynowego, których uczenie wykonywane jest przed zobaczeniem jakiejkolwiek gry. Należałoby potraktować to bardziej jako spójny system niż „sumę jego części", być może umożliwiając dodatkową naukę w czasie gry.

W planach jest wystawienie własnego rozwiązania w kolejnej edycji zawodów, które to mają potencjał stanowić motywację i czynnik napędzający dalszy rozwój badań w dziedzinie.